Fabrice Kordon
Janos Sztipanovits (Eds.)

LNCS 4322

# Reliable Systems on Unreliable Networked Platforms

12th Monterey Workshop 2005
Laguna Beach, CA, USA, September 2005
Revised Selected Papers

Springer

Fabrice Kordon Janos Sztipanovits (Eds.)

# Reliable Systems on Unreliable Networked Platforms

12th Monterey Workshop 2005
Laguna Beach, CA, USA, September 22-24, 2005
Revised Selected Papers

Springer

Volume Editors

Fabrice Kordon
Université Pierre et Marie Curie
Laboratoire d'Informatique de Paris 6
104 Avenue du Président Kennedy, 75016 Paris, France
E-mail: Fabrice.Kordon@lip6.fr

Janos Sztipanovits
Vanderbilt University
School of Engineering
Nashville, TN 37235-6306, USA
E-mail: janos.sztipanovits@vanderbilt.edu

Library of Congress Control Number: 2007921535

CR Subject Classification (1998): D.1.3, D.2-3, D.4.5, F.3, C.2.1, C.2-4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-71155-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-71155-1 Springer Berlin Heidelberg New York



Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12026487 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4322

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

# Preface

## Networked Systems: Realization of Reliable Systems on Unreliable Networked Platforms

The Monterey Workshops series was initiated in 1993 by David Hislop with the purpose of exploring the critical problems associated with cost-effective development of high-quality software systems. During its 12-year history, the Monterey Workshops have brought together scientists that share a common interest in software development research serving practical advances in next-generation software-intensive systems. Each year is dedicated to a given topic such as "Software Engineering Tools: Compatibility and Integration" (Vienna in 2004), "Engineering for Embedded Systems: From Requirements to Implementation" (Chicago in 2003), "Radical Innovations of Software and Systems Engineering in the Future" (Venice in 2002), "Engineering Automation for Software Intensive System Integration" (Monterey in 2001), etc.

This $12^{th}$ Monterey Workshop was held in Laguna Beach, CA during September 22–24, 2005.

## Context of the $12^{th}$ Workshop

Networked computing is increasingly becoming the universal integrator for large-scale systems. In addition, new generations of wireless networked embedded systems rapidly create new technological environments that imply complex interdependencies amongst all layers of societal-scale critical infrastructure, such as transportation, energy distribution and telecommunication. This trend makes reliability and safety of networked computing a crucial issue and a technical precondition for building software-intensive systems that are robust, fault tolerant, and highly available.

The $12^{th}$ Monterey Workshop on "Networked Systems: Realization of Reliable Systems on Unreliable Networked Platforms" focused on new, promising directions in achieving high software and system reliability in networked systems.

All presentations at the workshop were by invitation upon the advice of the Program Committee.

### Invited Speakers

| | |
|---|---|
| Myla Archer | Naval Research Lab, USA |
| Barrett Bryant | University of Alabama, Birmingham, USA |
| David Corman | Boeing, St Louis, USA |
| Nick Dutt | UCI, USA |
| Holger Giese | University of Paderborn, Germany |
| Chris Gill | Washington University at St Louis, USA |
| Helen Gill | NSF, USA |
| Klaus Havelund | NASA, USA |

| | |
|---|---|
| David Hislop | Army Research Office, USA |
| Liviu Iftode | Rutgers, USA |
| Vana Kalogeraki | UC Riverside, USA |
| Gabor Karsai | Vanderbilt University, USA |
| Kane Kim | University of California at Irvine, USA |
| Moon-Hae Kim | Konkuk University, Korea |
| Raymond Klefstad | UCI, USA |
| Hermann Kopetz | Vienna University of Technology, Austria |
| Fabrice Kordon | University of Pierre & Marie Curie, Paris, France |
| Ingolf Krueger | UCSD, USA |
| Akos Ledeczi | Vanderbilt University, USA |
| Edward Lee | UC Berkeley (Keynote Presentation), USA |
| Chenyang Lu | Washington University, USA |
| Luqi | Naval Postgraduate School, USA |
| Zohar Manna | Stanford University, USA |
| Oliver Marin | University of Pierre & Marie Curie, Paris, France |
| Nenad Medvidovic | USC, USA |
| Laurent Pautet | Télécom Paris, France |
| Raj Rajkumar | Carnegie Mellon University, USA |
| Martin Rinard, | MIT, USA |
| Man-tak Shing | Naval Postgraduate School, USA |
| Janos Sztipanovits | Vanderbilt University, USA |
| Wei-Tek Tsai | Arizona State University, USA |
| Andre Van der Hoek | UCI, USA |
| Nalini Venkatasubramanian | UCI, USA |
| Ben Watson | Lockheed Martin, USA |
| Albert Wavering | NIST, USA |
| Victor Winter | University of Nebraska at Omaha, USA |
| Feng Zhao | Microsoft Research (Keynote Presentation), USA |

Papers included in this volume were selected among the submissions from the workshop's discussions.

## Workshop Topics

Software is the new infrastructure of the information age. It is fundamental to economic success, scientific and technical research and national security. Our current ability to construct the large and complex software systems demanded for continued economic progress is inadequate.

The workshop discussed a range of challenges in networked systems that require further major advances in software and systems technology:

- **System Integration and Dynamic Adaptation.** A new challenge in networked systems is that stable application performance needs to be maintained in spite of the dynamically changing communication and computing platforms. Consequently, the run-time architecture must include active control mechanisms for adapting the

system/software components to changing conditions. Global system characteristics need to be achieved by increased run-time use of reflection (systems that utilize their own models), advanced interface modeling, self-adaptation, and self-optimization.

- **Effects of Dynamic Structure.** The structure of networked systems is complex and highly dynamic. Because systems are formed by ad hoc networks of nodes and connections, they lack fine-grain determinism for end-to-end behaviors that span subsystem and network boundaries. In addition, there are end-to-end system qualities such as timeliness and security that can only be evaluated in this dynamically integrated context.
- **Effects of Faults.** Faults and disruptions in the underlying communication and computing infrastructure are the normal events. Since well-understood techniques for fault-tolerant computing, such as n-modular redundancy, are not applicable in the dynamically changing networked architecture, new technology is required for building safe and reliable applications on dynamic, distributed platforms.
- **Design for Reliability.** Although there are varieties of metrics and established practices for characterizing the expected failure behavior of a system after it is fielded and there are established practices for specifying the desired reliability of a system, the evaluation of system or software reliability prior to fielding is a significant problem.
- **System Certification.** The process for certifying that a system meets specified reliability goals under the range of conditions expected in actual use currently involves exhaustive analysis of a system, including its development history and extensive testing. Current methods do not give systems engineers the confidence they would like to have in concluding that a system will have particular reliability characteristics.
- **Effects of Scale.** Another risk that overlays all proposed solutions is scale. Scale also addresses both run-time and design-time concerns. Typically, demonstrations are the convincing drivers to technology adoption. Demonstrations of new technologies however are usually small-scale, focused efforts. It is an open problem how to scale up a demonstration that addresses the number of nodes and connections, and the number of software developers, analysts, and integrators to provide enough proof to justify technology transition.

These challenges are exaggerated in networked-embedded software systems, where computation and communication are tightly integrated with physical process.

## Approaches

There have been important new developments during the past five years that improve our chance to meet the new challenges listed above. Contributions at the workshop identified and discussed research approaches that have direct and immediate relevance to the new challenges. Listed below are the major themes that came up in many forms in the presentations and captured in the contributions of these proceedings.

*Model-based software development of network-centric system-of-systems.* Model-based design is rapidly becoming one of the prominent software/system development paradigms. Models with precisely defined syntax and semantics capture system/software invariants that can be formally analyzed and used for predicting/ verifying system behavior and for generating code. A new challenge in network-centric system-of-systems is that design invariants need to be maintained actively during run-time due to the dynamically changing communication and computing platforms. Consequently, the relationship between design-time modeling and model analysis and run-time behavior needs to be fundamentally different: emphasis needs to be shifted toward correct-by-construction approaches that can guarantee selected behavioral properties without the need for system-level verification, and the run-time architecture must include active control for adapting the system/software to changing conditions. Global system characteristics need to be achieved by increased run-time use of reflection (systems that utilize their own models), advanced interface modeling, self-adaptation, and self-optimization.

*Foundations of future design and programming abstractions.* Programming abstractions have a crucial role in the design of highly concurrent, dynamic, and time-critical networked systems. Today's abstractions have been developed for programs with static structure, closed architectures, and stable computing platforms that are not scalable, understandable, and analyzable in complex, networked, real-time systems. We need abstractions that go beyond a narrow view of programming languages to integrate modeling, design, and analysis. They must satisfy the need for blending solid formal foundations with domain-specific expressions and must yield behavior that is predictable and understandable to system designers, even in the face of uncertain or dynamic system structure. To accomplish this, they must serve both the modeling role and the design role, leveraging generators, visual notations, formal semantics, probabilistic modeling, and yet-to-be-developed techniques for gaining an effective multiplicity of views into a design. And they must effectively express concurrency, quality-of-service constraints, and heterogeneity.

*Active fault management in network-centric systems.* It is important to recognize that software will never be perfect large-scale, networked systems-of-systems. Software and platform components may fail at any time. The notion of active fault management accepts this as a fact and instead of attempting to mask the faults, it focuses on their containment, mitigation, and management. Active fault management is a novel technique that is gaining acceptance in complex engineering systems (e.g., aerospace vehicles) and promises reliability through detecting, isolating and recovering from faults using algorithmic techniques for contingency management. The software engineering community took notice of these engineering techniques and applies them to software artifacts. The resulting fault management architectures are layered, as different methods may be needed on different levels of abstractions in systems and, preferably, they have to be proactive, so that they detect early precursors to larger problems (e.g., memory leak in dynamically allocated memory, or memory fragmentation) such that the system will have sufficient time to take preventive action.

*Intelligent, robust middleware.* Complexity of large-scale networked systems requires careful consideration on reusability of code. Middleware technologies offer architec-

tural solutions for separating application code from highly reusable components or layers in software stacks. We need to develop and validate a new generation of intelligent middleware technologies that can adapt dependably in response to dynamically changing conditions for the purpose of always utilizing the available computer and network infrastructure to the highest degree possible in support of system needs. Emerging architectures, such as service-oriented architecture (SOA), provide focus for this new generation of middleware research that will ultimately enable software whose functional and QoS-related properties can be modified either statically, (e.g., to reduce footprint, leverage capabilities that exist in specific platforms, enable functional subsetting, and minimize hardware/software infrastructure dependencies) or dynamically (e.g., to optimize system responses to changing environments or requirements, such as changing component interconnections, power-levels, CPU/network bandwidth, latency/jitter, and dependability needs).

*Model-based development of certifiable systems.* Systems that are safety certified are arguably some of the most costly to develop. As a result, software architectures for such systems are typically very deterministic in order to enable provable mitigation of safety hazards. The limitations of these approaches are quickly becoming unacceptable due to the advent of ad-hoc mobile networks requiring a much more dynamic structure and expected unavailability of certain resources for these safety critical systems. Model-based development approaches must be applied to enable the development of these systems within reasonable cost. These approaches should include the development of modeling syntax and semantics to express safety-critical aspects and perhaps constrain dynamism, the provision of design-time and run-time analysis that leverages this model and addresses the concerns of the safety community in the context of a network-centric system of systems, the automatic generation of artifacts that are proven by analysis to be safe, and the establishment of trust in such tools and techniques by the safety community as a whole.

## Acknowledgement

We are grateful to the Steering Committee, the Local Organizing Committee and the invited speakers for making the workshop a success. We gratefully acknowledge sponsorship from the Army Research Office (David Hislop) and from the National Science Foundation (Helen Gill).

January 2007

Fabrice Kordon
Janos Sztipanovits

# Organization

## Executive Committee

| | |
|---|---|
| Conference Chair: | Kane Kim (University of California, Irvine, USA) |
| Program Chairs: | Fabrice Kordon (Université Pierre & Marie Curie, France) |
| | Janos Sztipanovits (Vanderbilt University, USA) |

## Technical Program Committee

| | |
|---|---|
| Carlos Delgado Kloos | University Carlos III of Madrid, Spain |
| Bertil Folliot | University of Pierre & Marie Curie, Paris, France |
| Tom Henzinger | Ecole Polytechnique Federale de Lausanne, Switzerland |
| Kane Kim | University of California at Irvine, USA |
| Insup Lee | University of Pennsylvania, USA |
| Chenyang Lu | Washington University, USA |
| Tom Maibaum | King's College, London, UK |
| Ugo Montanari | University of Pisa, Italy |
| Laurent Pautet | Télécom Paris, France |
| Wolfgang Pree | University of Salzburg, Austria |
| Doug Schmidt | Vanderbilt University - ISIS, USA |

# Lecture Notes in Computer Science

For information about Vols. 1–4302

please contact your bookseller or Springer

Vol. 4429: R. Lu, J.H. Siekmann, C. Ullrich (Eds.), Cognitive Systems. X, 161 pages. 2007. (Sublibrary LNAI).

Vol. 4405: L. Padgham, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VII. XII, 225 pages. 2007.

Vol. 4403: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Evolutionary Multi-Criterion Optimization. XIX, 954 pages. 2007.

Vol. 4397: C. Stephanidis, M. Pieper (Eds.), Universal Access in Ambient Intelligence Environments. XV, 467 pages. 2007.

Vol. 4396: J. García-Vidal, L. Cerdà-Alabern (Eds.), Wireless Systems and Mobility in Next Generation Internet. IX, 271 pages. 2007.

Vol. 4394: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing. XVI, 648 pages. 2007.

Vol. 4393: W. Thomas, P. Weil (Eds.), STACS 2007. XVIII, 708 pages. 2007.

Vol. 4392: S.P. Vadhan (Ed.), Theory of Cryptography. XI, 595 pages. 2007.

Vol. 4390: S.O. Kuznetsov, S. Schmidt (Eds.), Formal Concept Analysis. X, 329 pages. 2007. (Sublibrary LNAI).

Vol. 4385: K. Coninx, K. Luyten, K.A. Schneider (Eds.), Task Models and Diagrams for Users Interface Design. XI, 355 pages. 2007.

Vol. 4384: T. Washio, K. Satoh, H. Takeda, A. Inokuchi (Eds.), New Frontiers in Artificial Intelligence. IX, 401 pages. 2007. (Sublibrary LNAI).

Vol. 4383: E. Bin, A. Ziv, S. Ur (Eds.), Hardware and Software, Verification and Testing. XII, 235 pages. 2007.

Vol. 4381: J. Akiyama, W.Y.C. Chen, M. Kano, X. Li, Q. Yu (Eds.), Discrete Geometry, Combinatorics and Graph Theory. XI, 289 pages. 2007.

Vol. 4380: S. Spaccapietra, P. Atzeni, F. Fages, M.-S. Hacid, M. Kifer, J. Mylopoulos, B. Pernici, P. Shvaiko, J. Trujillo, I. Zaihrayeu (Eds.), Journal on Data Semantics VIII. XV, 219 pages. 2007.

Vol. 4378: I. Virbitskaite, A. Voronkov (Eds.), Perspectives of Systems Informatics. XIV, 496 pages. 2007.

Vol. 4377: M. Abe (Ed.), Topics in Cryptology – CT-RSA 2007. XI, 403 pages. 2006.

Vol. 4376: E. Frachtenberg, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing. VII, 257 pages. 2007.

Vol. 4373: K. Langendoen, T. Voigt (Eds.), Wireless Sensor Networks. XIII, 358 pages. 2007.

Vol. 4372: M. Kaufmann, D. Wagner (Eds.), Graph Drawing. XIV, 454 pages. 2007.

Vol. 4371: K. Inoue, K. Satoh, F. Toni (Eds.), Computational Logic in Multi-Agent Systems. X, 315 pages. 2007. (Sublibrary LNAI).

Vol. 4370: P.P Lévy, B. Le Grand, F. Poulet, M. Soto, L. Darago, L. Toubiana, J.-F. Vibert (Eds.), Pixelization Paradigm. XV, 279 pages. 2007.

Vol. 4369: M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel, O. Takata (Eds.), Declarative Programming for Knowledge Management. X, 229 pages. 2006. (Sublibrary LNAI).

Vol. 4368: T. Erlebach, C. Kaklamanis (Eds.), Approximation and Online Algorithms. X, 345 pages. 2007.

Vol. 4367: K. De Bosschere, D. Kaeli, P. Stenström, D. Whalley, T. Ungerer (Eds.), High Performance Embedded Architectures and Compilers. XI, 307 pages. 2007.

Vol. 4366: K. Tuyls, R. Westra, Y. Saeys, A. Nowé (Eds.), Knowledge Discovery and Emergent Complexity in Bioinformatics. IX, 183 pages. 2007. (Sublibrary LNBI).

Vol. 4364: T. Kühne (Ed.), Models in Software Engineering. XI, 332 pages. 2007.

Vol. 4362: J. van Leeuwen, G.F. Italiano, W. van der Hoek, C. Meinel, H. Sack, F. Plášil (Eds.), SOFSEM 2007: Theory and Practice of Computer Science. XXI, 937 pages. 2007.

Vol. 4361: H.J. Hoogeboom, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. IX, 555 pages. 2006.

Vol. 4360: W. Dubitzky, A. Schuster, P.M.A. Sloot, M. Schroeder, M. Romberg (Eds.), Distributed, High-Performance and Grid Computing in Computational Biology. X, 192 pages. 2007. (Sublibrary LNBI).

Vol. 4358: R. Vidal, A. Heyden, Y. Ma (Eds.), Dynamical Vision. IX, 329 pages. 2007.

Vol. 4357: L. Buttyán, V. Gligor, D. Westhoff (Eds.), Security and Privacy in Ad-Hoc and Sensor Networks. X, 193 pages. 2006.

Vol. 4355: J. Julliand, O. Kouchnarenko (Eds.), B 2007: Formal Specification and Development in B. XIII, 293 pages. 2006.

Vol. 4354: M. Hanus (Ed.), Practical Aspects of Declarative Languages. X, 335 pages. 2006.

Vol. 4353: T. Schwentick, D. Suciu (Eds.), Database Theory – ICDT 2007. XI, 419 pages. 2006.

Vol. 4352: T.-J. Cham, J. Cai, C. Dorai, D. Rajan, T.-S. Chua, L.-T. Chia (Eds.), Advances in Multimedia Modeling, Part II. XVIII, 743 pages. 2006.

Vol. 4351: T.-J. Cham, J. Cai, C. Dorai, D. Rajan, T.-S. Chua, L.-T. Chia (Eds.), Advances in Multimedia Modeling, Part I. XIX, 797 pages. 2006.

Vol. 4349: B. Cook, A. Podelski (Eds.), Verification, Model Checking, and Abstract Interpretation. XI, 395 pages. 2007.

Vol. 4348: S.T. Taft, R.A. Duff, R.L. Brukardt, E. Ploedereder, P. Leroy (Eds.), Ada 2005 Reference Manual. XXII, 765 pages. 2006.

Vol. 4347: J. Lopez (Ed.), Critical Information Infrastructures Security. X, 286 pages. 2006.

Vol. 4346: L. Brim, B. Haverkort, M. Leucker, J. van de Pol (Eds.), Formal Methods: Applications and Technology. X, 363 pages. 2007.

Vol. 4345: N. Maglaveras, I. Chouvarda, V. Koutkias, R. Brause (Eds.), Biological and Medical Data Analysis. XIII, 496 pages. 2006. (Sublibrary LNBI).

Vol. 4344: V. Gruhn, F. Oquendo (Eds.), Software Architecture. X, 245 pages. 2006.

Vol. 4342: H. de Swart, E. Orłowska, G. Schmidt, M. Roubens (Eds.), Theory and Applications of Relational Structures as Knowledge Instruments II. X, 373 pages. 2006. (Sublibrary LNAI).

Vol. 4341: P.Q. Nguyen (Ed.), Progress in Cryptology - VIETCRYPT 2006. XI, 385 pages. 2006.

Vol. 4340: R. Prodan, T. Fahringer, Grid Computing. XXIII, 317 pages. 2007.

Vol. 4339: E. Ayguadé, G. Baumgartner, J. Ramanujam, P. Sadayappan (Eds.), Languages and Compilers for Parallel Computing. XI, 476 pages. 2006.

Vol. 4338: P. Kalra, S. Peleg (Eds.), Computer Vision, Graphics and Image Processing. XV, 965 pages. 2006.

Vol. 4337: S. Arun-Kumar, N. Garg (Eds.), FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science. XIII, 430 pages. 2006.

Vol. 4335: S.A. Brueckner, S. Hassas, M. Jelasity, D. Yamins (Eds.), Engineering Self-Organising Systems. XII, 212 pages. 2007. (Sublibrary LNAI).

Vol. 4334: B. Beckert, R. Hähnle, P.H. Schmitt (Eds.), Verification of Object-Oriented Software. XXIX, 658 pages. 2007. (Sublibrary LNAI).

Vol. 4333: U. Reimer, D. Karagiannis (Eds.), Practical Aspects of Knowledge Management. XII, 338 pages. 2006. (Sublibrary LNAI).

Vol. 4332: A. Bagchi, V. Atluri (Eds.), Information Systems Security. XV, 382 pages. 2006.

Vol. 4331: G. Min, B. Di Martino, L.T. Yang, M. Guo, G. Ruenger (Eds.), Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops. XXXVII, 1141 pages. 2006.

Vol. 4330: M. Guo, L.T. Yang, B. Di Martino, H.P. Zima, J. Dongarra, F. Tang (Eds.), Parallel and Distributed Processing and Applications. XVIII, 953 pages. 2006.

Vol. 4329: R. Barua, T. Lange (Eds.), Progress in Cryptology - INDOCRYPT 2006. X, 454 pages. 2006.

Vol. 4328: D. Penkler, M. Reitenspiess, F. Tam (Eds.), Service Availability. X, 289 pages. 2006.

Vol. 4327: M. Baldoni, U. Endriss (Eds.), Declarative Agent Languages and Technologies IV. VIII, 257 pages. 2006. (Sublibrary LNAI).

Vol. 4326: S. Göbel, R. Malkewitz, I. Iurgel (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. X, 384 pages. 2006.

Vol. 4325: J. Cao, I. Stojmenovic, X. Jia, S.K. Das (Eds.), Mobile Ad-hoc and Sensor Networks. XIX, 887 pages. 2006.

Vol. 4323: G. Doherty, A. Blandford (Eds.), Interactive Systems. XI, 269 pages. 2007.

Vol. 4322: F. Kordon, J. Sztipanovits (Eds.), Reliable Systems on Unreliable Networked Platforms. XIV, 317 pages. 2007.

Vol. 4320: R. Gotzhein, R. Reed (Eds.), System Analysis and Modeling: Language Profiles. X, 229 pages. 2006.

Vol. 4319: L.-W. Chang, W.-N. Lie (Eds.), Advances in Image and Video Technology. XXVI, 1347 pages. 2006.

Vol. 4318: H. Lipmaa, M. Yung, D. Lin (Eds.), Information Security and Cryptology. XI, 305 pages. 2006.

Vol. 4317: S.K. Madria, K.T. Claypool, R. Kannan, P. Uppuluri, M.M. Gore (Eds.), Distributed Computing and Internet Technology. XIX, 466 pages. 2006.

Vol. 4316: M.M. Dalkilic, S. Kim, J. Yang (Eds.), Data Mining and Bioinformatics. VIII, 197 pages. 2006. (Sublibrary LNBI).

Vol. 4314: C. Freksa, M. Kohlhase, K. Schill (Eds.), KI 2006: Advances in Artificial Intelligence. XII, 458 pages. 2007. (Sublibrary LNAI).

Vol. 4313: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods. IX, 197 pages. 2006.

Vol. 4312: S. Sugimoto, J. Hunter, A. Rauber, A. Morishima (Eds.), Digital Libraries: Achievements, Challenges and Opportunities. XVIII, 571 pages. 2006.

Vol. 4311: K. Cho, P. Jacquet (Eds.), Technologies for Advanced Heterogeneous Networks II. XI, 253 pages. 2006.

Vol. 4310: T. Boyanov, S. Dimova, K. Georgiev, G. Nikolov (Eds.), Numerical Methods and Applications. XIII, 715 pages. 2007.

Vol. 4309: P. Inverardi, M. Jazayeri (Eds.), Software Engineering Education in the Modern Age. VIII, 207 pages. 2006.

Vol. 4308: S. Chaudhuri, S.R. Das, H.S. Paul, S. Tirthapura (Eds.), Distributed Computing and Networking. XIX, 608 pages. 2006.

Vol. 4307: P. Ning, S. Qing, N. Li (Eds.), Information and Communications Security. XIV, 558 pages. 2006.

Vol. 4306: Y. Avrithis, Y. Kompatsiaris, S. Staab, N.E. O'Connor (Eds.), Semantic Multimedia. XII, 241 pages. 2006.

Vol. 4305: A.A. Shvartsman (Ed.), Principles of Distributed Systems. XIII, 441 pages. 2006.

Vol. 4304: A. Sattar, B.-H. Kang (Eds.), AI 2006: Advances in Artificial Intelligence. XXVII, 1303 pages. 2006. (Sublibrary LNAI).

Vol. 4303: A. Hoffmann, B.-H. Kang, D. Richards, S. Tsumoto (Eds.), Advances in Knowledge Acquisition and Management. XI, 259 pages. 2006. (Sublibrary LNAI).

# Table of Contents

# Reinventing Computing for Real Time

Edward A. Lee and Yang Zhao

University of California, Berkeley
{eal,ellen_zh}@eecs.berkeley.edu

**Abstract.** This paper studies models of computation, software techniques, and analytical models for distributed timed systems. By "timed systems" we mean those where timeliness is an essential part of the behavior. By "distributed systems" we mean computational systems that are interconnected on a network. Applications of timed distributed systems include industrial automation, distributed immersive environments, advanced instrumentation systems, networked control systems, and many modern embedded software systems that integrate networking. The introduction of network time protocols such as NTP (at a coarse granularity) and IEEE 1588 (at a fine granularity) makes possible time coherence that has not traditionally been part of the computational models in networked systems. The main question we address in this paper is: Given time synchronization with some known precision, how does this change how distributed applications are designed and developed? A second question we address is: How can time synchronization help with realizing coordinated real-time events.

## 1 Introduction

Despite considerable progress in software and hardware techniques, when embedded computing systems absolutely must meet tight timing constraints, many of the advances in computing become part of the problem, not part of the solution. Although synchronous digital logic delivers precise timing determinacy, advances in computer architecture and software have made it difficult or impossible to estimate or predict the execution time of software. Moreover, networking techniques introduce variability and stochastic behavior, and operating systems rely on best effort techniques. Worse, programming languages lack time in their semantics, so timing requirements are only specified indirectly. This paper studies methods for programming ensembles of networked real-time, embedded computers where time and concurrency are first-class properties of the program.

This contrasts with established software techniques, where time and concurrency are afterthoughts. The prevailing view of real-time appears to have been established well before embedded computing was common. Wirth reduces real-time programming to threads with bounds on execution time, arguing that "it is prudent to extend the conceptual framework of sequential programming as little as possible and, in particular, to avoid the notion of execution time" [30]. In this sequential framework, "computation" is accomplished by a terminating sequence of state transformations. This core abstraction underlies the design of nearly all

F. Kordon and J. Sztipanovits (Eds.): Monterey Workshop 2005, LNCS 4322, pp. 1–25, 2007.

computers, programming languages, and operating systems in use today. But unfortunately, this core abstraction does not fit embedded software very well.

This core abstraction fits reasonably well if embedded software is simply "software on small computers." In this view, embedded software differs from other software only in its resource limitations (small memory, small data word sizes, and relatively slow clocks). In this view, the "embedded software problem" is an optimization problem. Solutions emphasize efficiency; engineers write software at a very low level (in assembly code or C), avoid operating systems with a rich suite of services, and use specialized computer architectures such as programmable DSPs and network processors that provide hardware support for common operations. These solutions have defined the practice of embedded software design and development for the last 25 years or so. In an analysis that remains as valid today as 18 years ago, Stankovic laments the resulting misconceptions that real-time computing "is equivalent to fast computing" or "is performance engineering" [29].

Of course, thanks to the semiconductor industry's ability to follow Moore's law, the resource limitations of 25 years ago should have almost entirely evaporated today. Why then has embedded software design and development changed so little? It may be that extreme competitive pressure in products based on embedded software, such as consumer electronics, rewards only the most efficient solutions. This argument is questionable, however. There are many examples where functionality has proven more important than efficiency. It is arguable that resource limitations are not the only defining factor for embedded software, and may not even be the principal factor.

Stankovic argues that "the time dimension must be elevated to a central principle of the system. Time requirements and properties cannot be an afterthought" [29]. But in mainstream computing, this has not happened. The "time dimension," of course, is inextricably linked to concurrency, and prevailing models of concurrency (threads and message passing) are in fact obstacles to elevating time to a central principle.

In embedded software, several recent innovations provide unconventional ways of programming concurrent and/or timed systems. We point to six cases that define concurrency models, component architectures, and management of time-critical operations in ways significantly different from prevailing software engineering techniques. The first is nesC with TinyOS [8], which was developed for programming very small programmable sensor nodes called "motes." The second is Click [16], which was created to support the design of software-based network routers. These first two have an imperative flavor, and components interact principally through procedure calls. The third is Simulink with Real-Time Workshop (from The MathWorks), which was created for embedded control software and is widely used in the automotive industry. The fourth is SCADE (from Esterel Technologies, see [2], which was created for safety-critical embedded software and is used in avionics. These two have a more declarative flavor, where components interact principally through messages rather than procedure calls. The fifth is the family of hardware description languages, including Verilog, VHDL, and

SystemC, which express vast amounts of concurrency, principally using discrete-event semantics. The sixth example is LabVIEW, from National Instruments, a dataflow programming environment with a visual syntax designed for embedded instrumentation applications. The amount and variety of experimentation with alternative models of computation for embedded systems is yet a further indication that the prevailing software abstractions are inadequate.

The approach in this paper leverages the concept of actor-oriented design [20], borrowing ideas from Simulink and from Giotto [12], an experimental real-time programming language. However, it addresses a number of limitations in Simulink and Giotto by building similar multitasking implementations from specifications that combine dataflow modeling and distributed discrete-event modeling. In discrete-event models, components interact with one another via events that are placed on a time line. Some level of agreement about time across distributed components is necessary for this model to have a coherent semantics. While distribution of discrete-event models has long been used to exploit parallel computing to accelerate execution [31], we are not concerned here with accelerating execution. The focus is instead on using a model of time as a binding coordination agent. This steers us away from conservative techniques (like Chandy and Misra [3]) and optimistic techniques (like Time Warp [15]). One interesting possibility is based on distributed consensus (as in Croquet [28]). In this paper, we focus on techniques based on distributing discrete-event models, with functionality specified by dataflow models. Our technique allows out of order execution without sacrificing determinacy and without requiring backtracking. The use of dataflow formalisms [26] supports mixing untimed and event-triggered computation with timed and periodic computation.

## 2 Embedded Software

There are clues that embedded software differs from other software in quite fundamental ways. If we examine carefully why engineers write embedded software in assembly code or C, we discover that efficiency is not the only concern, and may not even be the main concern. The reasons may include, for example, the need to count cycles in a critical inner loop, not to make it fast, but rather to make it predictable. No widely used programming language integrates a way to specify timing requirements or constraints. Instead, the abstractions they offer are about scalability (inheritance, dynamic binding, polymorphism, memory management), and if anything further obscure timing (consider the impact of garbage collection on timing). Counting cycles, of course, becomes extremely difficult on modern processor architectures, where memory hierarchy (caches), dynamic dispatch, and speculative execution make it nearly impossible to tell how long it will take to execute a particular piece of code. Embedded software designers may choose alternative processor architectures such as programmable DSPs not only for efficiency reasons, but also for predictability of timing.

Another reason engineers stick to low-level programming is that embedded software has to interact with hardware that is specialized to the application.